Java Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking

1. Sharing resources
2. Centralize software management

Do You Know ?

* How to perform connection-oriented Socket Programming in networking ?
* How to display the data of any online web page ?
* How to get the IP address of any host name e.g. www.google.com ?
* How to perform connection-less socket programming in networking ?

The java.net package supports two protocols,

1. **TCP:** Transmission Control Protocol provides reliable communication between the sender and receiver. TCP is used along with the Internet Protocol referred as TCP/IP.
2. **UDP:** User Datagram Protocol provides a connection-less protocol service by allowing packet of data to be transferred along two or more nodes

Java Networking Terminology

The widely used Java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

* TCP
* FTP
* Telnet
* SMTP
* POP etc.

3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

4) MAC Address

MAC (Media Access Control) address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC address.

For example, an ethernet card may have a **MAC** address of 00:0d:83::b1:c0:8e.

5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

6) Socket

A socket is an endpoint between two way communications.

Visit next page for Java socket programming.

java.net package

The java.net package can be divided into two sections:

1. **A Low-Level API:** It deals with the abstractions of addresses i.e. networking identifiers, Sockets i.e. bidirectional data communication mechanism and Interfaces i.e. network interfaces.
2. **A High Level API:** It deals with the abstraction of URIs i.e. Universal Resource Identifier, URLs i.e. Universal Resource Locator, and Connections i.e. connections to the resource pointed by URLs.

The java.net package provides many classes to deal with networking applications in Java. A list of these classes is given below:

* Authenticator
* CacheRequest
* CacheResponse
* ContentHandler
* CookieHandler
* CookieManager
* DatagramPacket
* DatagramSocket
* DatagramSocketImpl
* InterfaceAddress
* JarURLConnection
* MulticastSocket
* InetSocketAddress
* InetAddress
* Inet4Address
* Inet6Address
* IDN
* HttpURLConnection
* HttpCookie
* NetPermission
* NetworkInterface
* PasswordAuthentication
* Proxy
* ProxySelector
* ResponseCache
* SecureCacheResponse
* ServerSocket
* Socket
* SocketAddress
* SocketImpl
* SocketPermission
* StandardSocketOptions
* URI
* URL
* URLClassLoader
* URLConnection
* URLDecoder
* URLEncoder
* URLStreamHandler

**List of interfaces available in java.net package:**

* ContentHandlerFactory
* CookiePolicy
* CookieStore
* DatagramSocketImplFactory
* FileNameMap
* SocketOption<T>
* SocketOptions
* SocketImplFactory
* URLStreamHandlerFactory
* ProtocolFamily

# Java Socket Programming

Java Socket programming is used for communication between the applications running on different JRE.

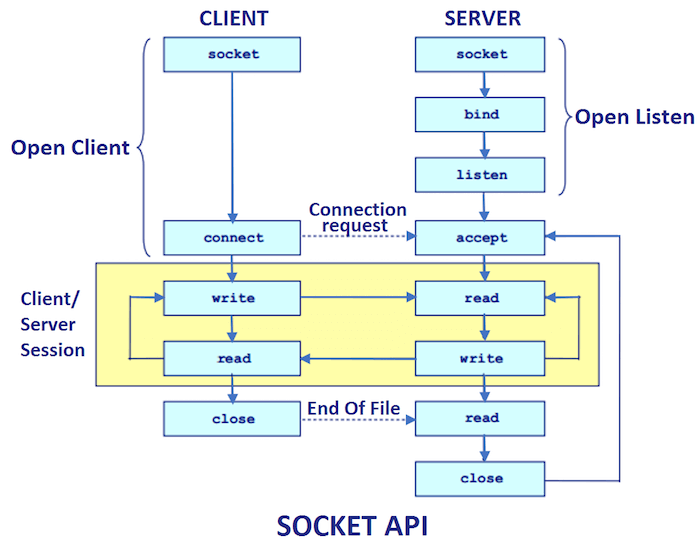
Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.



## Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

### Important methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public InputStream getInputStream() | returns the InputStream attached with this socket. |
| 2) public OutputStream getOutputStream() | returns the OutputStream attached with this socket. |
| 3) public synchronized void close() | closes this socket |

## ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

### Important methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public Socket accept() | returns the socket and establish a connection between server and client. |
| 2) public synchronized void close() | closes the server socket. |

## Example of Java Socket Programming

**Creating Server:**

To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6666 port number for the communication between the client and server. You may also choose any other port number. The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.

1. ServerSocket ss=**new** ServerSocket(6666);
2. Socket s=ss.accept();//establishes connection and waits for the client

**Creating Client:**

To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.

1. Socket s=**new** Socket("localhost",6666);

Let's see a simple of Java socket programming where client sends a text and server receives and prints it.

*File: MyServer.java*

1. **import** java.io.\*;
2. **import** java.net.\*;
3. **public** **class** MyServer {
4. **public** **static** **void** main(String[] args){
5. **try**{
6. ServerSocket ss=**new** ServerSocket(6666);
7. Socket s=ss.accept();//establishes connection
8. DataInputStream dis=**new** DataInputStream(s.getInputStream());
9. String  str=(String)dis.readUTF();
10. System.out.println("message= "+str);
11. ss.close();
12. }**catch**(Exception e){System.out.println(e);}
13. }
14. }

*File: MyClient.java*

1. **import** java.io.\*;
2. **import** java.net.\*;
3. **public** **class** MyClient {
4. **public** **static** **void** main(String[] args) {
5. **try**{
6. Socket s=**new** Socket("localhost",6666);
7. DataOutputStream dout=**new** DataOutputStream(s.getOutputStream());
8. dout.writeUTF("Hello Server");
9. dout.flush();
10. dout.close();
11. s.close();
12. }**catch**(Exception e){System.out.println(e);}
13. }
14. }

[download this example](https://www.javatpoint.com/src/networking/socket.zip)

To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure.

After running the client application, a message will be displayed on the server console.



## Example of Java Socket Programming (Read-Write both side)

In this example, client will write first to the server then server will receive and print the text. Then server will write to the client and client will receive and print the text. The step goes on.

*File: MyServer.java*

1. **import** java.net.\*;
2. **import** java.io.\*;
3. **class** MyServer{
4. **public** **static** **void** main(String args[])**throws** Exception{
5. ServerSocket ss=**new** ServerSocket(3333);
6. Socket s=ss.accept();
7. DataInputStream din=**new** DataInputStream(s.getInputStream());
8. DataOutputStream dout=**new** DataOutputStream(s.getOutputStream());
9. BufferedReader br=**new** BufferedReader(**new** InputStreamReader(System.in));
11. String str="",str2="";
12. **while**(!str.equals("stop")){
13. str=din.readUTF();
14. System.out.println("client says: "+str);
15. str2=br.readLine();
16. dout.writeUTF(str2);
17. dout.flush();
18. }
19. din.close();
20. s.close();
21. ss.close();
22. }}

*File: MyClient.java*

1. **import** java.net.\*;
2. **import** java.io.\*;
3. **class** MyClient{
4. **public** **static** **void** main(String args[])**throws** Exception{
5. Socket s=**new** Socket("localhost",3333);
6. DataInputStream din=**new** DataInputStream(s.getInputStream());
7. DataOutputStream dout=**new** DataOutputStream(s.getOutputStream());
8. BufferedReader br=**new** BufferedReader(**new** InputStreamReader(System.in));
10. String str="",str2="";
11. **while**(!str.equals("stop")){
12. str=br.readLine();
13. dout.writeUTF(str);
14. dout.flush();
15. str2=din.readUTF();
16. System.out.println("Server says: "+str2);
17. }
19. dout.close();
20. s.close();
21. }}

# Java URL

The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example:

1. https://www.javatpoint.com/java-tutorial



A URL contains many information:

1. **Protocol:** In this case, http is the protocol.
2. **Server name or IP Address:** In this case, www.javatpoint.com is the server name.
3. **Port Number:** It is an optional attribute. If we write http//ww.javatpoint.com:80/sonoojaiswal/ , 80 is the port number. If port number is not mentioned in the URL, it returns -1.
4. **File Name or directory name:** In this case, index.jsp is the file name.

## Constructors of Java URL class

**URL(String spec)**

Creates an instance of a URL from the String representation.

**URL(String protocol, String host, int port, String file)**

Creates an instance of a URL from the given protocol, host, port number, and file.

**URL(String protocol, String host, int port, String file, URLStreamHandler handler)**

Creates an instance of a URL from the given protocol, host, port number, file, and handler.

**URL(String protocol, String host, String file)**

Creates an instance of a URL from the given protocol name, host name, and file name.

**URL(URL context, String spec)**

Creates an instance of a URL by parsing the given spec within a specified context.

**URL(URL context, String spec, URLStreamHandler handler)**

Creates an instance of a URL by parsing the given spec with the specified handler within a given context.

## Commonly used methods of Java URL class

The java.net.URL class provides many methods. The important methods of URL class are given below.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public String getProtocol() | it returns the protocol of the URL. |
| public String getHost() | it returns the host name of the URL. |
| public String getPort() | it returns the Port Number of the URL. |
| public String getFile() | it returns the file name of the URL. |
| public String getAuthority() | it returns the authority of the URL. |
| public String toString() | it returns the string representation of the URL. |
| public String getQuery() | it returns the query string of the URL. |
| public String getDefaultPort() | it returns the default port of the URL. |
| public URLConnection openConnection() | it returns the instance of URLConnection i.e. associated with this URL. |
| public boolean equals(Object obj) | it compares the URL with the given object. |
| public Object getContent() | it returns the content of the URL. |
| public String getRef() | it returns the anchor or reference of the URL. |
| public URI toURI() | it returns a URI of the URL. |

## Example of Java URL class

1. //URLDemo.java
2. **import** java.net.\*;
3. **public** **class** URLDemo{
4. **public** **static** **void** main(String[] args){
5. **try**{
6. URL url=**new** URL("http://www.javatpoint.com/java-tutorial");
8. System.out.println("Protocol: "+url.getProtocol());
9. System.out.println("Host Name: "+url.getHost());
10. System.out.println("Port Number: "+url.getPort());
11. System.out.println("File Name: "+url.getFile());
13. }**catch**(Exception e){System.out.println(e);}
14. }
15. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=URLDemo)

Output:

Protocol: http

Host Name: www.javatpoint.com

Port Number: -1

File Name: /java-tutorial

Let us see another example URL class in Java.

1. //URLDemo.java
2. **import** java.net.\*;
3. **public** **class** URLDemo{
4. **public** **static** **void** main(String[] args){
5. **try**{
6. URL url=**new** URL("https://www.google.com/search?q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8");
8. System.out.println("Protocol: "+url.getProtocol());
9. System.out.println("Host Name: "+url.getHost());
10. System.out.println("Port Number: "+url.getPort());
11. System.out.println("Default Port Number: "+url.getDefaultPort());
12. System.out.println("Query String: "+url.getQuery());
13. System.out.println("Path: "+url.getPath());
14. System.out.println("File: "+url.getFile());
16. }**catch**(Exception e){System.out.println(e);}
17. }
18. }

Output:

Protocol: https

Host Name: www.google.com

Port Number: -1

Default Port Number: 443

Query String: q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8

Path: /search

File: /search?q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8

# Java URLConnection Class

The **Java URLConnection** class represents a communication link between the URL and the application. It can be used to read and write data to the specified resource referred by the URL.

### What is the URL?

* URL is an abbreviation for Uniform Resource Locator. An URL is a form of string that helps to find a resource on the World Wide Web (WWW).
* URL has two components:

1. The protocol required to access the resource.
2. The location of the resource.

### Features of URLConnection class

1. URLConnection is an abstract class. The two subclasses HttpURLConnection and JarURLConnection makes the connetion between the client Java program and URL resource on the internet.
2. With the help of URLConnection class, a user can read and write to and from any resource referenced by an URL object.
3. Once a connection is established and the Java program has an URLConnection object, we can use it to read or write or get further information like content length, etc.

### Constructors

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| 1) protected URLConnection(URL url) | It constructs a URL connection to the specified URL. |

### URLConnection Class Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| void addRequestProperty(String key, String value) | It adds a general request property specified by a key-value pair |
| void connect() | It opens a communications link to the resource referenced by this URL, if such a connection has not already been established. |
| boolean getAllowUserInteraction() | It returns the value of the allowUserInteraction field for the object. |
| int getConnectionTimeout() | It returns setting for connect timeout. |
| Object getContent() | It retrieves the contents of the URL connection. |
| Object getContent(Class[] classes) | It retrieves the contents of the URL connection. |
| String getContentEncoding() | It returns the value of the content-encoding header field. |
| int getContentLength() | It returns the value of the content-length header field. |
| long getContentLengthLong() | It returns the value of the content-length header field as long. |
| String getContentType() | It returns the value of the date header field. |
| long getDate() | It returns the value of the date header field. |
| static boolean getDefaultAllowUserInteraction() | It returns the default value of the allowUserInteraction field. |
| boolean getDefaultUseCaches() | It returns the default value of an URLConnetion's useCaches flag. |
| boolean getDoInput() | It returns the value of the URLConnection's doInput flag. |
| boolean getDoInput() | It returns the value of the URLConnection's doOutput flag. |
| long getExpiration() | It returns the value of the expires header files. |
| static FileNameMap getFilenameMap() | It loads the filename map from a data file. |
| String getHeaderField(int n) | It returns the value of nth header field |
| String getHeaderField(String name) | It returns the value of the named header field. |
| long getHeaderFieldDate(String name, long Default) | It returns the value of the named field parsed as a number. |
| int getHeaderFieldInt(String name, int Default) | It returns the value of the named field parsed as a number. |
| String getHeaderFieldKey(int n) | It returns the key for the nthheader field. |
| long getHeaderFieldLong(String name, long Default) | It returns the value of the named field parsed as a number. |
| Map<String, List<String>> getHeaderFields() | It returns the unmodifiable Map of the header field. |
| long getIfModifiedSince() | It returns the value of the object's ifModifiedSince field. |
| InputStream getInputStream() | It returns an input stream that reads from the open condition. |
| long getLastModified() | It returns the value of the last-modified header field. |
| OutputStream getOutputStream() | It returns an output stream that writes to the connection. |
| Permission getPermission() | It returns a permission object representing the permission necessary to make the connection represented by the object. |
| int getReadTimeout() | It returns setting for read timeout. |
| Map<String, List<String>> getRequestProperties() | It returns the value of the named general request property for the connection. |
| URL getURL() | It returns the value of the URLConnection's URL field. |
| boolean getUseCaches() | It returns the value of the URLConnection's useCaches field. |
| Static String guessContentTypeFromName(String fname) | It tries to determine the content type of an object, based on the specified **file** component of a URL. |
| static String guessContentTypeFromStream(InputStream is) | It tries to determine the type of an input stream based on the characters at the beginning of the input stream. |
| void setAllowUserInteraction(boolean allowuserinteraction) | It sets the value of the allowUserInteraction field of this URLConnection. |
| static void setContentHandlerFactory(ContentHandlerFactory fac) | It sets the ContentHandlerFactory of an application. |
| static void setDefaultAllowUserInteraction(boolean defaultallowuserinteraction) | It sets the default value of the allowUserInteraction field for all future URLConnection objects to the specified value. |
| void steDafaultUseCaches(boolean defaultusecaches) | It sets the default value of the useCaches field to the specified value. |
| void setDoInput(boolean doinput) | It sets the value of the doInput field for this URLConnection to the specified value. |
| void setDoOutput(boolean dooutput) | It sets the value of the doOutput field for the URLConnection to the specified value. |

## How to get the object of URLConnection Class

The openConnection() method of the URL class returns the object of URLConnection class.

**Syntax:**

1. **public** URLConnection openConnection()**throws** IOException{}

## Displaying Source Code of a Webpage by URLConnecton Class

The URLConnection class provides many methods. We can display all the data of a webpage by using the getInputStream() method. It returns all the data of the specified URL in the stream that can be read and displayed.

### Example of Java URLConnection Class

1. **import** java.io.\*;
2. **import** java.net.\*;
3. **public** **class** URLConnectionExample {
4. **public** **static** **void** main(String[] args){
5. **try**{
6. URL url=**new** URL("http://www.javatpoint.com/java-tutorial");
7. URLConnection urlcon=url.openConnection();
8. InputStream stream=urlcon.getInputStream();
9. **int** i;
10. **while**((i=stream.read())!=-1){
11. System.out.print((**char**)i);
12. }
13. }**catch**(Exception e){System.out.println(e);}
14. }
15. }